Planning with Multi-Constraints via Collaborative Language Agents

Cong Zhang*, **Derrick Goh Xin Deik***, **Dexun Li, Hao Zhang, and Yong Liu**
Huawei Noah's Ark Lab
cong.zhang92@gmail.com, goh.xin.deik@huawei.com, lidexun@huawei.com,
zhang.hao3@huawei.com, liu.yong6@huawei.com

Abstract

The rapid advancement of neural language models has sparked a new surge of intelligent agent research. Unlike traditional agents, large language model-based agents (LLM agents) have emerged as a promising paradigm for achieving artificial general intelligence (AGI) due to their superior reasoning and generalization capabilities. Effective planning is crucial for the success of LLM agents in real-world tasks, making it a highly pursued topic in the community. Current planning methods typically translate tasks into executable action sequences. However, determining a feasible or optimal sequence for complex tasks with multiple constraints at fine granularity, which often requires compositing long chains of heterogeneous actions, remains challenging. This paper introduces Planning with Multi-Constraints (PMC), a **zero-shot** methodology for collaborative LLM-based multi-agent systems that simplifies complex task planning with constraints by decomposing it into a hierarchy of subordinate tasks. Each subtask is then mapped into executable actions. PMC was assessed on two constraint-intensive benchmarks, TravelPlanner and API-Bank. Notably, PMC achieved an average 42.68% success rate on TravelPlanner, significantly higher than GPT-4 (2.92%), and outperforming GPT-4 with ReAct on API-Bank by 13.64%, showing the immense potential of integrating LLM with multi-agent systems. We also show that PMC works with small LLM as the planning core, e.g., LLaMA-3.1-8B. Our code is publically available at https://github.com/zcaicaros/PMC.

1 Introduction

Recently, there has been a growing interest in using large language models (LLMs) as the cognitive core of agents (Wang et al., 2024b), due to their ability to understand and execute human instructions in natural language. LLM-powered agents, known for their strong logical skills and strategic planning, are considered a promising path toward achieving artificial general intelligence (AGI) (Wang et al., 2024b; You et al., 2024).

Current LLM agent planning solutions aim to map tasks to sequences of executable actions (Huang et al., 2024). The *plan-then-execute* methods (Shen et al., 2024; Wang et al., 2023a) break down complex tasks into small, manageable sub-tasks to facilitate the inference of a sequence of executable actions. In contrast, the *step-by-step* methods (Wei et al., 2022; Yao et al., 2023b; Chen et al., 2023; Wu et al., 2023; Gao et al., 2023b) interleave planning and execution, where each action is determined based on previous outcomes. The former simply assumes each sub-task can be executed with a single tool (Shen et al., 2024), but real-world applications often require tools with diverse functionalities (Krishnakumar and Sheth, 1995). The latter is unsuitable for time-sensitive constraints requiring comprehensive condition assessment, meticulous planning and subsequent execution. Moreover, the piecemeal nature of these approaches may lead to suboptimal outcomes and potential task failure. To improve planning stability and performance, recent studies (Dagan et al., 2023; Guan et al., 2023; Yang et al., 2023c) integrate LLMs with external planning tools requiring task descriptions in specific formats, *e.g.,* first-order logic (Barwise, 1977). However, translating various tasks into certain computational logic can be challenging and often demands a myriad of domain knowledge (Cropper and Dumančić, 2022). Existing LLM-based multi-agent systems primarily simulate human behaviors and social activities (Li et al., 2024; Park et al., 2023; Gao et al., 2023a), while planning for collaborative multi-agent systems under multiple constraints, despite their significant potential, remains underexplored.

In this work, we propose **P**lanning with **M**ulti-**C**onstraints (PMC), a *zero-shot* planning method

*Both authors contributed equally to this work.

Proceedings of the 31st International Conference on Computational Linguistics, pages 10054–10082
January 19–24, 2025.

for collaborative LLM-based multi-agent systems. PMC simplifies complex task planning by breaking it down into a hierarchy of subordinate tasks, each achievable through a series of (heterogeneous) tool calls. Specifically, PMC comprises a *manager* agent for task decomposition and a fleet of *executor* agents to perform sub-tasks. The manager performs *task-level planning* by decomposing the task into a graph, where each node represents a specific sub-task (e.g., recommendation) and the edges delineate the dependency topology among tasks. Then each sub-task is decomposed into a sequence of function calls, *i.e., step-level planning and execute*, via an executor. The executor may utilize off-the-shelf planning techniques, like ReAct (Yao et al., 2023b), to facilitate sub-task accomplishment. PMC can be viewed as a framework that extends the capabilities of individual LLM agents by equipping with cooperation cores, thus transforming them into collaborative multi-agent system. PMC focuses on complex planning scenarios with constraints, such as budget limitations, which are categorized into "local" and "global" types. Local constraints are managed by executors during sub-task execution, whereas global constraints are considered in conjunction with other variables. To improve the success rate and stability, PMC employs a *supervisor* agent to refine a sub-task if the results of the previous sub-tasks it depends on are obtained and a *deliverer* agent to produce the final outcome. If the given sub-task does not depend on any previous sub-task, the supervisor agent will not refine the sub-task. The deliverer agent will either summarize the outcome or make a decision for the user based on the result of all sub-tasks.

Distinct from the toy tasks (Singh et al., 2023) or puzzles (Ahn et al., 2024) commonly used in existing planning methods, we evaluate PMC on two real-world applications: itinerary planning and daily tool usage. Experiment results show that PMC achieves substantial performance gains on two benchmarks. Specifically, PMC obtains 42.68% success rate on TravelPlanner (Xie et al., 2024), a significant increase from GPT-4 (2.92%). It also surpasses GPT-4 with ReAct on API-Bank (Li et al., 2023) by 13.62% in absolute improvement. To the best of our knowledge, PMC is the first plan-and-execute method for collaborative LLM-based multi-agent systems to effectively address complex tasks involving multiple constraints. Moreover, Ww show that LLaMA-3.1-8B equipped with PMC under one demonstration example surpasses GPT-4 by a large margin.

2 Literature Review for Language Model Agent Planning

The emergence of LLMs introduces new paradigms for agents (Chu et al., 2023; Wang et al., 2024a; Masterman et al., 2024), demonstrating significant intelligence in reasoning (Kojima et al., 2022; Wei et al., 2023; Wang et al., 2023b), planning (Yao et al., 2023b,a; Besta et al., 2024), instruction-following (Xu et al., 2023; Wang et al., 2023c; Ren et al., 2023), and tool-usage (Schick et al., 2023; Yang et al., 2023b; Shen et al., 2024) across various domains. Planning acts as an essential capability to interact with external environments, which involves organizing thought trajectories, setting objectives, and determining steps to accomplish the objectives (Mattar and Lengyel, 2022). Some work (Wei et al., 2023; Yao et al., 2023b; Chen et al., 2023; Wang et al., 2023a) focuses on task decomposition, aiming to solve complex tasks in a divide-and-conquer manner. The plan selection methods (Yao et al., 2023a; Besta et al., 2024; Wang et al., 2023b; Xiao and Wang, 2023) elicit LLMs to generate various alternative plans for a task following by a search algorithm for optimal plan selection and execution. Recent studies (Shinn et al., 2023; Madaan et al., 2023; Huang et al., 2022; Gou et al., 2024) also explore to enhance LLM's planning ability via reflection and refinement strategies. Moreover, some work (Liu et al., 2023a; Lin et al., 2023; Zhao et al., 2024) also introduces external planners to aid the planning procedure of LLMs.

Numerous strategies have been developed to harness the potentials of LLMs for specific agent planning (Xi et al., 2023), whose effectiveness and accuracy of planning significantly determine the agent's robustness and usability. Web-agents (Yao et al., 2022; Deng et al., 2023; Gur et al., 2024; Furuta et al., 2024) explore the interaction between LLM and web-environment by simulating human's web-browsing behaviors via RL-based planning or trajectory planning. General tool-agents require to interact with massive APIs or tools, making the planning procedures more challenging. Solutions to tool-agent planning usually rely on various task decomposition (Yuan et al., 2024; Shen et al., 2024), self-rectification (Ma et al., 2024) and domain-reasoning (Lu et al., 2023) strategies. Other task-specific agents focus on designing sophisticated

planning strategies, such as tree search (Zhou et al., 2023) and Bayesian adaptive MDPs (Liu et al., 2023b). Multi-agent systems (Chen et al., 2024; Hong et al., 2024; Gong et al., 2023; Mei et al., 2024) seek to solve more complex real-world tasks by combining multiple powerful LLM-based agents. Existing solutions mainly focus on tackling the complexities inherent in integrating heterogeneous agents with different capabilities and specializations (Mei et al., 2024), while the planning strategies among these agents are overlooked. In contrast, our PMC focuses on designing generalized, robust planning strategies for multi-agent systems. Although $\texttt{LLM}_{api}$Swarm (Zhuge et al., 2024) shares a similar concept with PMC, it focuses on visualizing multi-agent collaboration via composite graphs to aid prompt tuning, while our PMC is a planning algorithm specifically designed for systems with multiple collaborative agents.

3 Preliminaries

The LLM-based agent is an AI system utilizing an LLM as its computational core, enabling functionalities beyond text generation such as task execution, logical reasoning, and autonomous operation. Formally, an LLM agent includes: $A = (\texttt{LLM}, \mathcal{F}_n, R, \mathcal{S}, C)$. $\texttt{LLM}$ is the language model instance (e.g., LLaMA (Touvron et al., 2023)) used for reasoning, planning, and decision-making. $\mathcal{F}_n$ is a set of functions/actions performed by the agent. R is the agent's role as defined by the prompt. $S \in \mathcal{S}$ represents the agent's dynamic state, including knowledge and internal processes. C is the communication module for exchanging information with other agents or the environment. In a collaborative LLM-based multi-agent system, multiple agents, $[A_1, A_2, \ldots, A_m]$, work together to achieve a common goal. Each agent A_i has a specific role R_i and task specialization, *e.g.,* task decomposition.

4 Planning with Multi-Constraints

Complex projects, such as those in construction or manufacturing, often present significant challenges, particularly with diverse, geographically dispersed teams. However, thorough planning, effective communication, and collaboration can ensure successful outcomes (Lester, 2017). In light of this, we introduce planning with multi-constraints (PMC), a novel zero-shot planning approach for collaborative LLM-based multi-agent systems to address complex tasks with multiple constraints. In PMC, a designated *manager* agent decomposes a complex task $\mathcal{T} = \{T_i | i \in S(K) = \{1, \cdots, K\}\}$ into smaller, more manageable sub-tasks, $\{T_1, T_2, T_3, \cdots, T_K\}$ through *task-level planning*. These sub-tasks are then converted into a sequence of heterogeneous tool-using actions executed by a fleet of *executor* agents, a process referred to as *step-level planning and execution*. Additionally, PMC incorporates a supervisor agent to facilitate sharing synthesized sub-task outcomes among executors and a deliverer agent to consolidate final results upon the collective findings of all sub-tasks. The comprehensive framework of PMC is illustrated in Figure 1.

4.1 The Collaborative Multi-Agent System Design

The description of each agent is shown in each subsection and the prompt structure of each agent is illustrated in Figure 2.

4.1.1 Manager Agent

The manager agent has two primary objectives. Firstly, it decomposes the intricate task $\mathcal{T}$ into a set of interconnected sub-tasks $\mathcal{T} = \{T_i | i \in S(K) = \{1, \cdots, K\}\}$. These sub-tasks often exhibit dependencies, where completing one task is contingent on completing another. For instance, deciding on hotels usually depends on finalizing the trip destination. Thus, the manager must identify and define these inter-dependencies meticulously. Additionally, the manager has to assign suitable executors to each sub-task. Executors are viewed as a collection of composite tools, and the manager matches them based on the sub-task requirements, a method termed the *executor as tools* technique.

Secondly, the manager must make well-informed decisions on task assignments, especially under constraints like budget limits or specific transportation needs in travel scenarios. Some constraints can be managed during individual sub-tasks. For instance, for a sub-task that searching for accommodation, the minimum stay requirement ensures only suitable hotels are considered. The manager also needs to identify constraints that interact with other variables across sub-tasks and cannot be solved within a single sub-task. For instance, when selecting a flight, available hotel and restaurant options provided by other sub-tasks must be jointly considered. Identifying local and global constraints to *divide and conquer* them is crucial for successfully completing complex tasks. The formal definitions

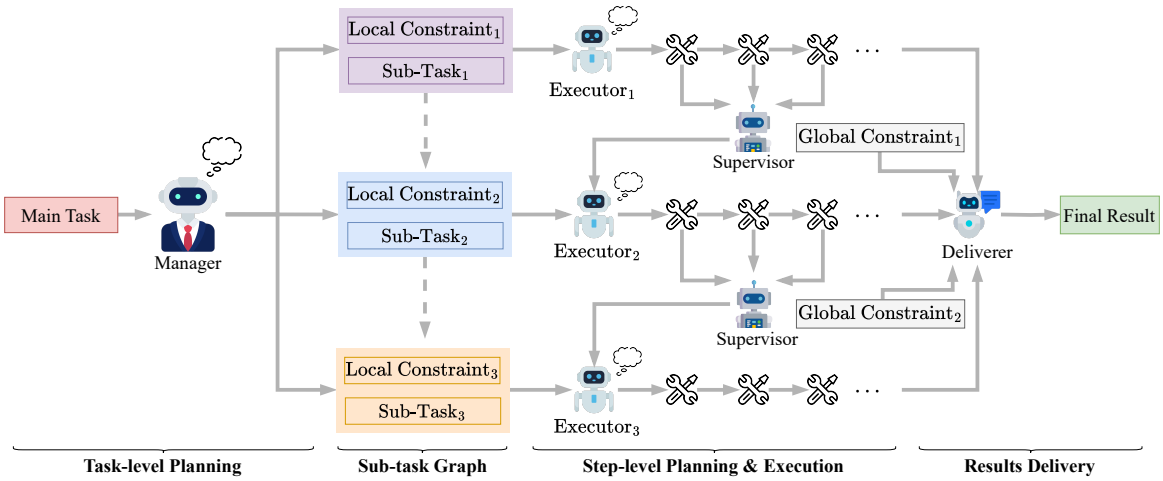

Figure 1: An overview of PMC Framework. The PMC Framework provides a structured methodology for managing and executing sub-tasks within a directed sub-task graph topology, as the manager coordinates. For instance, the completion of Task$_2$ depends on the outputs derived from Task$_1$, which a supervisor agent subsequently consolidates. The executor agent is tasked with implementing the sub-task, considering any local constraints present. Upon completion of the sub-tasks, the deliverer agent is responsible for aggregating all sub-task outcomes to satisfy global constraints and subsequently achieve the overarching task objectives.

```
# Role
...
# Output Format
...
# Instructions, e.g., rules, etc.
<for meta-tasks... >
<for constraints... >
...
# Executors as Tools
...
```

(a) Manager.

```
# Role
...
# Instructions, e.g., rules, etc.
...
# Tools
...
# Output Format
...
```

(b) Executors.

```
# Role
...
# Output Format
...
# Instructions, e.g., rules, etc.
...
```

(c) Supervisor.

```
# Role
...
# Output Format
...
# Instructions, e.g., rules, etc.
...
```

(d) Deliverer.

Figure 2: The zero-shot prompt structure for each agent. Note both supervisor and deliverer agents do not require function calls, while different executors will have different tool lists.

for the local and global constraints are presented as follows:

Definition 4.1. A constraint C_l is *local* if and only if $\exists!\ i \in S(K)$ such that C_l can be fulfilled purely based on the results of T_i.

Definition 4.2. A constraint C_g is *global* if and only if $\exists\ \mathcal{T}_{C_g} = \{T_i | i \in S(K)\} \subseteq S(K)$ and $K \geq |\mathcal{T}_c| > 1$, such that C_g can be fulfilled based on the results of all $T_i \in \mathcal{T}_{C_g}$, where $|\cdot|$ demotes the cardinality of a set.

It is important to note that the manager identifies potential constraints and categorizes them into local and global ones exclusively based on internal knowledge. No prior information about the constraints for $\mathcal{T}$ is provided, ensuring that the zero-shot property of PMC is maintained. Figure 2(a) depicts the logic for manager prompt design.

4.1.2 Executor Agent

The manager agent assigns each sub-task to an executor agent, which has access to various heterogeneous tools (*e.g.,* functions). The executor aims to create a sequence of actions (*e.g.,* function calls) to complete the assigned sub-task while adhering to local constraint C_l. This process significantly reduces the planning complexity, as the executor focuses on a specific, well-defined task with clear requirements and constraints. Consequently, applying off-the-shelf single-agent planning methods to map a task to an execution sequence is feasible. Figure 2(b) illustrates the conceptual prompt design of the executor. Owing to the functional variation among executors, a tailored design approach is necessary for each executor, depending on the specific tools available to them.

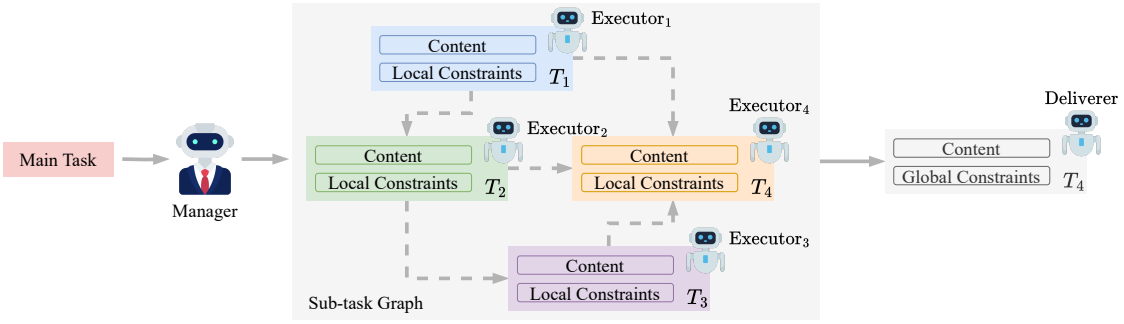

Figure 3: An overview of sub-task graph, which reveals the task-level decomposition. The manager agent decomposes the main task into several sub-tasks with inter-dependencies (dashed arrows).

4.1.3 Supervisor Agent

The role of the supervisor agent is to refine the sub-task T_i by incorporating synthesized outcomes from neighboring sub-tasks. After the manager agent decomposes the main task, only ambiguous objectives (*e.g.,* "*Finding a hotel in city B*") and inter-dependencies among sub-tasks (*e.g.,* "*Searching flight to New York*" $\rightarrow$ "*Finding a hotel in city B*") are identified. To execute T_i effectively, its input parameters need precise specifications. For instance, the input "*Finding a hotel in city B*" must be correctly instantiated as "*Finding a hotel in New York near John F. Kennedy International Airport*" based on outcomes (*e.g.,* "*Booked flight ZC9896 to New York, arriving at John F. Kennedy International Airport*") from preceding sub-task (*e.g.,* "*Searching flight to New York*"). To address these nuances and eliminate ambiguities, the supervisor agent acts before the commencement of T_i. It rewrites T_i by referencing the outcomes of all neighboring sub-tasks. This ensures all necessary parameters are included and accurately instantiated. Here, the neighbor of T_i is defined as the collection of sub-tasks that have direct inter-dependencies with T_i. Formally, the neighborhood of T_i is defined as:

Definition 4.3. The neighbors $\mathcal{N}(T_i)$ of sub-task T_i is defined as $\{T_j | \forall j \in S(K), s.t., T_j \rightarrow T_i\}$.

Remark: An alternative idea is to include all precedent sub-tasks in outcome synthesis for T_i instead of just its immediate neighbors. While this seems reasonable, it can overwhelm the manager agent, especially given the input limitations of LLMs with many tasks. In contrast, our proposed approach focuses on immediate neighbors, maintaining manageability and avoiding such complexities. Moreover, our approach is capable of preserving all information through the "message-passing" mechanism, allowing correct results from precedent sub-tasks to be recursively propagated to T_i. The following proposition supports this assertion:

Proposition 4.4. *A sub-task T_i is accomplishable while adhering to local constraints if and only if all the sub-tasks within its direct neighborhood $\mathcal{N}(T_i)$ are accomplished with their respective local constraints maintained.*

The proof is in Appendix D. The supervisor prompt design is delineated in Figure 2(c).

4.1.4 Deliverer Agent

The primary objective of the deliverer agent is to synthesize the outcomes of all sub-tasks while ensuring alignment with the global constraints, $\{C_g\}$. This synthesis is critical because $\{C_g\}$ can only be effectively addressed once all sub-task results are available. Thus, the deliverer agent is uniquely positioned to manage these constraints, ensuring that the final results comprehensively satisfy all global constraints. The logic of the deliverer prompt design is depicted in Figure 2(d).

The detailed prompt design and technical introduction of all agents are presented in Appendix G. We provide the process of constructing and optimising the prompts in Appendix F. We hope this prompt construction process will give the community informative suggestions on how to build complex prompts from scratch.

4.2 Hierarchical Task Planning and Execution

Task-Level Planning. The manager agent will analyze the given complex task $\mathcal{T}$ and decompose it into a series of inter-dependent sub-tasks $\{T_1, T_2, \dots\}$. PMC represents them via a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, termed *sub-task graph*. In $\mathcal{G}$, each node $V_i \in \mathcal{V}$ corresponds to a sub-task T_i and

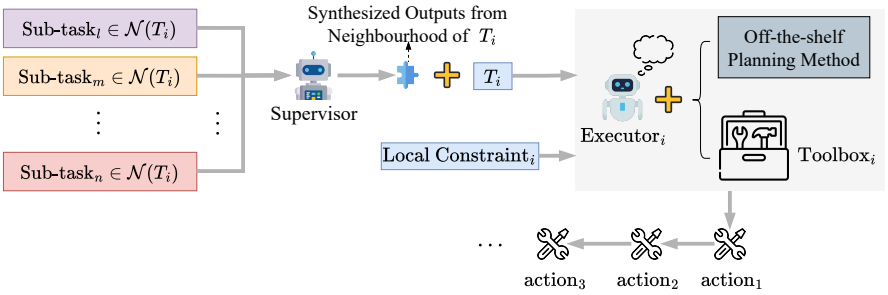

Figure 4: step-level Planning and Execution. The executor is furnished with a planning core and a toolbox comprising diverse functions. This includes an off-the-shelf planning algorithm such as ReAct (Yao et al., 2023b), which is used to translate the sub-task into a series of executable function calls required to accomplish the assigned sub-task.

With Unconventional Hint			GPT-3.5 + ReAct	GPT-4 + ReAct	GPT-3.5 + PMC	GPT-4 + PMC	GPT-4 (SP)
Validation Set (60)	Delivery Rate		98.33	98.33	91.67	96.67	100.00
	Common-sense	Micro	74.38	79.38	63.54	87.29	92.08
		Macro	0.00	8.33	1.67	43.33	50.00
	Hard Constraint	Micro	0.71	7.14	0.71	47.14	52.86
		Macro	0.00	5.00	1.67	46.67	28.33
	Final Pass Rate		0.00	1.67	0.00	**33.33**	13.33
Test Set (308)	Delivery Rate		93.50	98.38	84.09	97.40	100.00
	Common-sense	Micro	70.45	77.60	57.51	91.46	91.36
		Macro	0.32	7.46	1.95	50.00	45.45
	Hard Constraint	Micro	1.21	13.53	1.37	53.96	52.74
		Macro	0.32	9.74	1.30	45.12	29.22
	Final Pass Rate		0.00	2.92	0.65	**42.68**	14.94

Table 1: The Average Pass Rates (%) "**With** Unconventional Hint" for Instances Across All Difficulty Level on TravelPlanner. The highest final pass rates are highlighted in bold blue.

each edge $E_{ij} \in \mathcal{E}$ delineates the dependencies between sub-tasks T_i and T_j, where $i, j \in S(K)$. The architecture of the sub-task graph is illustrated in Figure 3. Executors adhere to the graph's topology, *i.e.,* its edge orientation, to ensure all prerequisites of a sub-task are met before its initiation, thereby enhancing the efficacy of the overall task execution. Furthermore, the sub-task graph serves as a tool for visualizing the task decomposition and an interactive interface to enhance the interpretability of PMC systems. It offers a mechanism for ongoing monitoring and potential human intervention, making it essential for PMC.

Step-Level Planning and Execution. After task-level decomposition, each sub-task is manageable to be further decomposed into a sequence of executable actions, *i.e.,* function calls. The complexity of each sub-task is significantly reduced as it is now a specific, well-defined task with clear requirements and local constraints, making the off-the-shelf planning method directly applicable, *e.g.,* ReAct (Yao et al., 2023b). Specifically, before the commencement of sub-task T_i, the supervisor agent will rewrite T_i by referencing the outcomes of all neighboring sub-tasks $\mathcal{N}(T_i)$. Then, the local constraints C_l for sub-task T_i identified by the manager will be given as the auxiliary information together with the refined T_i to the executor agent A_i, which will utilize the planning method, *e.g.,* ReAct, to accomplish T_i by decoding T_i into a sequence of actions. The whole process is illustrated in Figure 4.

5 Experiments

To assess PMC, we move beyond the existing planning methods that largely focus on simplistic tasks (Singh et al., 2023) or puzzles (Ahn et al., 2024) irrelevant to practical applications. Instead, we evaluate PMC through its application to real-world constraints intensive scenarios. Specifically, we examine its efficacy in the domains of itinerary planning (Xie et al., 2024) and daily tools using (Li et al., 2023). We optimized PMC's prompt for each benchmark exclusively on the validation set and apply the prompt directly to the test sets. For (Li et al., 2023), the training dataset was employed as a proxy for the validation set owing to the lack of a dedicated validation set. For all benchmarks, each

Without Unconventional Hint			GPT-3.5 + ReAct	GPT-4 + ReAct	GPT-3.5 + PMC	GPT-4 + PMC	GPT-4 (SP)	LLaMA-3.1-8B + GPT-4 + PMC
Validation Set (60)	Delivery Rate		100.00	98.33	88.33	100.00	100.00	70.00
	Common-sense	Micro	73.54	75.21	58.75	90.00	90.42	68.33
		Macro	0.00	3.33	0.00	41.67	35.00	28.33
	Hard Constraint	Micro	0.71	14.28	0.00	55.71	52.14	32.15
		Macro	1.67	13.33	0.00	48.33	25.00	25.00
	Final Pass Rate		0.00	0.00	0.00	**31.67**	6.67	15.00
Test Set (308)	Delivery Rate		98.38	96.43	80.19	98.05	100.00	65.26
	Common-sense	Micro	69.60	70.74	51.54	85.96	88.07	57.35
		Macro	0.00	2.92	0.65	29.55	32.79	21.43
	Hard Constraint	Micro	1.37	14.44	0.91	50.91	50.91	23.71
		Macro	0.65	9.09	0.32	46.10	30.84	19.16
	Final Pass Rate		0.00	0.65	0.32	**22.40**	12.66	12.66

Table 2: The Average Pass Rates (%) "**Without** Unconventional Hint" for Instances Across All Difficulty Level on TravelPlanner. The highest final pass rates are highlighted in bold blue. For the last column "LLaMA-3.1-8B + GPT-4 + PMC", we employ LLaMA-3.1-8B as the planner and GPT-4 as the executor, where LLaMA-3.1-8B is prompted with one-shot example.

Constraint Type	GPT4 + ReAct + CoT			Sole-planning			PMC		
	Easy	Medium	Hard	Easy	Medium	Hard	Easy	Medium	Hard
Commonsense Constraint									
Within Sandbox	32.79	23.08	21.95	90.16	88.46	82.93	70.49	75.96	70.73
Complete Information	81.97	86.54	86.59	100.00	100.00	100	79.51	78.85	79.27
Within Current City	95.08	95.19	90.24	97.54	95.19	98.78	97.54	95.19	97.56
Reasonable City Route	88.52	88.46	87.80	100.00	99.04	100	94.26	93.27	97.56
Diverse Restaurants	81.15	76.92	71.95	98.36	86.54	95.12	88.52	83.65	97.56
Diverse Attractions	99.18	98.08	96.34	100.00	100.00	100	97.54	95.19	100.00
Non-conf. Transportation	93.44	96.15	95.12	95.08	95.19	97.56	93.44	92.31	100.00
Miminum Nights Stay	57.38	67.31	43.90	62.30	53.85	53.66	95.90	93.27	89.02
Hard Constraint									
Budget	13.93	10.58	9.76	50.00	34.62	14.63	46.72	42.31	46.34
Room Rule	-	13.16	17.81	-	47.37	65.75	-	57.89	57.53
Cuisine	-	11.43	13.51	-	65.71	45.95	-	62.86	51.35
Room Type	-	19.35	14.29	-	74.19	77.78	-	54.84	53.97
Transportation	-	-	15.07	-	-	82.19	-	-	60.27
Final									
Final Pass Rate	4.10	4.10	2.44	30.33	5.77	3.66	**43.44**	**34.62**	**42.68**

Table 3: The Pass Rates Constraints for the "With Unconventional Hint" scenario for TravelPlanner. The "-" marks indicate the corresponding constraints are not applicable. The highest final pass rates are highlighted in bold blue for each difficulty level.

instance is executed only once without sampling, though multiple trials could potentially enhance performance. We will make the code publicly available upon acceptance.

5.1 Experiment Setup

5.1.1 Benchmarks

TravelPlanner (Xie et al., 2024). In `TravelPlanner`, users specify their origin, destination, and individual requirements. The benchmark assesses the ability of language agents to (1) efficiently gather necessary information using appropriate tools and (2) create practical, personalized travel plans for users. The plan is assessed using four main metrics: (1) delivery rate (a plan has to be delivered within **30 steps (including planning and execution)**), (2) commonsense constraint pass rate, (3) hard constraint pass rate, and (4) final pass rate (the rate for meeting all commonsense and hard constraints), which is the most important metric for evaluation. For (2) and (3), we define the "micro" pass rate as the ratio of passed constraints to total constraints and the "macro" pass rate as the ratio of plans passing all constraints to total plans.

The travel duration can be 3, 5, or 7 days. Due to budget constraints, we demonstrate that a 3-day dataset sufficiently justifies the effectiveness of PMC. The queries are categorized as `easy`, `medium` or `hard`.

However, we found that the benchmark includes odd rules as part of its evaluation. For instance, choosing the same restaurant multiple times throughout a trip breaches the `Diverse Restaurants` constraint, and selecting an airport as a meal location breaches the `Within Sandbox`

constraint. Yet, under normal circumstances, it's reasonable for a tourist to return to a favoured restaurant or dine at airport restaurants during their trip. To ensure that the agent recognizes these rules as part of commonsense knowledge, we provide specific guidance to the planning agents: the `Deliverer Agent` in PMC and the `Planner` in React and Sole-Planning. We term this setting as "**with unconventional hint**" (or with "hint" for short). To maintain the integrity of the experiment and stay true to the objectives of the original `TravelPlanner` paper, we conduct a separate experiment that excludes this external knowledge. This experiment still incorporates the less conventional rules used in both `Diverse Restaurants` and `Within Sandbox` settings. We term this setting as "**without unconventional hint**" (or without "hint" for short).

As our method consists of tool-use and planning (two-stage), we compare our method with the two-stage baseline, `ReAct` from (Xie et al., 2024) using GPT-3.5-Turbo and GPT-4-Turbo as language models. We also further compare our method to the best sole-planning baseline, Direct GPT4-Turbo, which has provided necessary information to the agent and only require agent to output the travel plan.

API-Bank (Li et al., 2023). `API-Bank` is a benchmark designed to evaluate the tool-use capabilities of large language models, focusing on APIs that are commonly used in everyday life, such as email. The benchmark includes three levels of difficulty, with Level 3 being the most challenging. We chose Level 3 for our experiment because it best assesses the planning abilities of the agent.

The benchmark assesses agents based on Accuracy and "**ROUGE**" (ROUGE-L) scores. The Accuracy metric gauges the correctness of API calls based on user queries, calculated as the proportion of correct API calls to total predictions. We modified this metric for a more consistent and fair assessment by defining Correctness as the ratio of unique correct API calls to total predictions. This adjustment addresses the tendency of some language models, like GPT-3.5 and GPT-4, to make repetitive correct API calls. The ROUGE-L score evaluates the responses generated from these API calls. Our experiments indicate that using this refined Accuracy metric results in lower baseline scores.

In addition to Correctness, we introduce the "**Completeness**" to better assess task execution. Correctness alone may not fully capture an agent's performance, as minimal API calls could artificially inflate scores. Completeness measures the ratio of unique, correct API calls to the total required API calls for the task, addressing the limitations of Correctness and ensuring a more accurate evaluation of the agent's effectiveness. We also introduce another metric named "**Tool Repeats**", which measures how often the model correctly calls an API after its initial use. A lower number of repeats indicates fewer unnecessary inferences, signifying a more efficient solution.

5.2 Result Analysis

5.2.1 Result Analysis for `TravelPlanner`

From Tables 1 and 2, it is evident that PMC significantly outperforms all baseline methods irrespective of the presence of unconventional hints. Notably, when hints are included, GPT4 enhanced by PMC achieves a superior average final pass rate of 42.68% across all difficulty levels, compared to a meagre 2.92% by baselines. This data underscores the potential of integrating large language models (LLMs) with multi-agent systems, marking it as a promising area for future research in LLM-based agent systems.

In the absence of hints, the setting replicates that described in (Xie et al., 2024), where the highest final pass rate for baseline models stands at 0.56%, consistent with the original study's findings. In this scenario, PMC significantly improves with an average final pass rate of 22.4%, surpassing the best-reported baseline result in (Xie et al., 2024).

Notably, the sub-task Planner (PMC) significantly outperforms the Standard Planner (SP) in settings that employ hints and those that do not. The SP operates purely as a decision-making framework in which all elements necessary for completing the itinerary, such as multiple choices for hotels, flights, and restaurants, are pre-supplied; thus, the SP agent merely selects the most suitable options from these pre-defined sets to construct the final itinerary. This renders SP a relatively simpler task compared to PMC and other benchmarks, which necessitate the searching and gathering of necessary elements prior to decision-making. Nonetheless, PMC achieves a superior final pass rate, a finding which may appear counter-intuitive yet can be elucidated as follows: PMC's exceptional performance is attributable to its effective deployment of a divide-and-conquer strategy in managing constraints. By resolving numerous local constraints

Model		GPT-4-1106-Preview	GPT-4-0613	GPT-3.5-Turbo-0125	GPT-3.5-Turbo-0613	LLaMA-3.1-8B + GPT-4
Correctness % (↑)	CoT	71.48	57.65	35.38	34.74	-
	No CoT	41.58	35.11	67.06	41.80	-
	PMC	82.63	**85.12**	74.13	67.81	67.58
Completeness % (↑)	CoT	47.76	46.12	28.16	30.20	-
	No CoT	34.29	32.24	23.27	20.82	-
	PMC	**64.08**	58.37	43.27	40.40	70.61
ROUGE (↑)	CoT	0.2641	0.2846	0.3085	0.2656	-
	No CoT	0.2507	0.2644	0.2346	0.2016	-
	PMC	**0.4053**	0.3839	0.3894	0.3754	0.4171
Tool Repeats (↓)	CoT	90	49	50	32	-
	No CoT	38	15	155	118	-
	PMC	16	23	**7**	65	64

Table 4: The Performance on API-Bank. The highest performance for each criteria is highlighted in bold blue. For the last column "LLaMA-3.1-8B + GPT-4", we employ LLaMA-3.1-8B as the planner and GPT-4 as the executor, where LLaMA-3.1-8B is prompted with one-shot example.

during the execution of sub-tasks, PMC considerably reduces the complexity that the agent encounters in formulating the ultimate itinerary plan. Table 3 presents the detailed pass rates for individual constraints, indicating that PMC significantly outperforms GPT4+ReAct+CoT in terms of pass rates across all constraints. However, GPT-3.5 is less effective than GPT4 when equipped with PMC, possibly due to less model performance. We have provided illustrative results for each difficulty level in Appendix H. We also present the detailed results for each difficulty level (easy, medium, hard) in Appendix E.

5.2.2 Result Analysis for API-Bank

Firstly, PMC significantly enhances the performance of both GPT4 and GPT-3.5 across all critical evaluation metrics. Compared to existing baselines, PMC consistently demonstrates superior performance. Notably, the best performance reported in the original paper (Li et al., 2023) achieved a 70% success rate, which our reimplementation slightly exceeds at 71.48%. Thus, PMC stands out by surpassing the top method referenced in (Li et al., 2023) by a substantial margin of at least 14%. Furthermore, PMC excels in other key areas such as task completeness, achieving an impressive 64.08%, and exhibits significantly fewer redundant tool interactions, with a count of just seven. This robust performance underscores PMC's potential in redefining the capabilities of advanced language models. We have provided illustrative results for each difficulty level in Appendix H.

5.2.3 Results for LLaMA-3.1-8B

In addition to utilising only GPT models on PMC, we also test the performance of PMC with open-source LLM with significantly lesser parameters, i.e., LLaMA-3.1-8B. Specifically, we use LLaMA-3.1-8B as the planner agent, with prompt consists of one-shot example and without unconventional hint setting. The results for TravelPlanner and API-Bank are given in the last column in Table 2 and Table 4.

From the results we demonstrate that even with small open-source LLM as the planner in PMC, its performance still surpasses GPT-4, with ReAct or CoT by a large margin. These results demonstrate that PMC is still effective for small LLMs.

6 Conclusion

This paper presents Planning with Multi-Constraints (PMC), an innovative zero-shot methodology for collaborative LLM-based multi-agent systems. PMC simplifies complex task planning by breaking it down into hierarchical sub-tasks, each mapped to executable actions. PMC was evaluated on two benchmarks, TravelPlanner and API-Bank. It achieved an average success rate of about 42% on TravelPlanner, a significant improvement from the initial 0.6%, and outperformed GPT-4 with ReAct by 14% on API-Bank. However, the current design still requires human input from executor agents. Enhancing PMC by enabling the manager agent to autonomously design prompts for executor agents could optimize executor creation, accelerating PMC's practical application. Future research should focus on developing more autonomous agents through advanced prompt optimization, as suggested in recent literature (Yang et al., 2023a). This approach promises to refine PMC's functionality and expand its applicability without human intervention, leading to more intelligent and self-sufficient multi-agent systems.

References

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*.

Jon Barwise. 1977. An introduction to first-order logic. In *Studies in Logic and the Foundations of Mathematics*, volume 90, pages 5–46. Elsevier.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, page 17682–17690. Association for the Advancement of Artificial Intelligence (AAAI).

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. 2023. A survey of chain of thought reasoning: Advances, frontiers and future. *ArXiv*, abs/2309.15402.

Andrew Cropper and Sebastijan Dumančić. 2022. Inductive logic programming at 30: a new introduction. *Journal of Artificial Intelligence Research*, 74:765–850.

Gautier Dagan, Frank Keller, and Alex Lascarides. 2023. Dynamic planning with a llm. *arXiv preprint arXiv:2308.06391*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36, pages 28091–28114. Curran Associates, Inc.

Hiroki Furuta, Yutaka Matsuo, Aleksandra Faust, and Izzeddin Gur. 2024. Exposing limitations of language model agents in sequential-task compositions on the web. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Chen Gao, Xiaochong Lan, Nian Li, Yuan Yuan, Jingtao Ding, Zhilun Zhou, Fengli Xu, and Yong Li. 2023a. Large language models empowered agent-based modeling and simulation: A survey and perspectives. *arXiv preprint arXiv:2312.11970*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.

Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, and Jianfeng Gao. 2023. Mindagent: Emergent gaming interaction. *ArXiv*, abs/2309.09971.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094.

Izzeddin Gur, Hiroki Furuta, Austin V Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A real-world webagent with planning, long context understanding, and program synthesis. In *The Twelfth International Conference on Learning Representations*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022. Inner monologue: Embodied reasoning through planning with language models. *ArXiv*, abs/2207.05608.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of llm agents: A survey. *ArXiv*, abs/2402.02716.

Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc.

Narayanan Krishnakumar and Amit Sheth. 1995. Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3:155–186.

Eur Ing Albert Lester. 2017. Chapter 20 - planning blocks and subdivision of blocks. In Eur Ing Albert Lester, editor, *Project Management, Planning and Control (Seventh Edition)*, seventh edition edition, pages 131–142. Butterworth-Heinemann.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2024. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. 2023. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. In *Advances in Neural Information Processing Systems*, volume 36, pages 23813–23825. Curran Associates, Inc.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023a. Llm+p: Empowering large language models with optimal planning proficiency. *ArXiv*, abs/2304.11477.

Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. 2023b. Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency. *ArXiv*, abs/2309.17382.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yubo Ma, Zhibin Gou, Junheng Hao, Ruochen Xu, Shuohang Wang, Liangming Pan, Yujiu Yang, Yixin Cao, Aixin Sun, Hany Awadalla, and Weizhu Chen. 2024. Sciagent: Tool-augmented language models for scientific reasoning. *ArXiv*, abs/2402.11451.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Tula Masterman, Sandi Besen, Mason Sawtell, and Alex Chao. 2024. The landscape of emerging ai agent architectures for reasoning, planning, and tool calling: A survey. *ArXiv*, abs/2404.11584.

Marcelo G. Mattar and Máté Lengyel. 2022. Planning in the brain. *Neuron*, 110(6):914–934.

Kai Mei, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. 2024. Aios: Llm agent operating system. *ArXiv*, abs/2403.16971.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22.

Jie Ren, Yao Zhao, Tu Vu, Peter J. Liu, and Balaji Lakshminarayanan. 2023. Self-evaluation improves selective generation in large language models. *ArXiv*, abs/2312.09300.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv*, abs/2302.04761.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *ArXiv*, abs/2303.11366.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024a. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6).

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024b. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):1–26.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023c. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The rise and potential of large language model based agents: A survey. *ArXiv*, abs/2309.07864.

Hengjia Xiao and Peng Wang. 2023. Llm a*: Human in the loop large language models enabled a* search for robotics. *ArXiv*, abs/2312.01797.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. In *Forty-first International Conference on Machine Learning*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *ArXiv*, abs/2304.12244.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023a. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023b. Gpt4tools: Teaching large language model to use tools via self-instruction. In *Advances in Neural Information Processing Systems*, volume 36, pages 71995–72007. Curran Associates, Inc.

Zhun Yang, Adam Ishay, and Joohyung Lee. 2023c. Coupling large language models with logic programming for robust and general reasoning from text. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757. Curran Associates, Inc.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Jiaxuan You, Ge Liu, Yunzhu Li, Song Han, and Dawn Song. 2024. How far are we from agi. In *ICLR 2024 Workshops*.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2024. Easytool: Enhancing llm-based agents with concise tool instruction. *ArXiv*, abs/2401.06201.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *ArXiv*, abs/2310.04406.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jurgen Schmidhuber. 2024. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*.

A Limitations

Despite the remarkable performance, the current architecture of executor agents in PMC still necessitates human input. There is an opportunity for significant enhancements within PMC by enabling the manager agent to autonomously generate the prompts for the executor agents, thereby optimizing the process of executor creation. Such advancements could notably accelerate the practical applications and enhance the efficacy of PMC.

B Social Impacts

This research explores the implementation of LLM-based agents to aid humans in solving complex tasks. While this automation promises increased productivity and focus on high-level tasks, it raises concerns about transparency and interpretability. The uncertainty of LLMs can obscure decision-making processes, potentially reducing trust in sensitive applications where understanding AI's reasoning is crucial. Additionally, by automating communication, there is a risk of diminishing important human interactions and nuances essential for quality exchanges and relationships in professional environments. It is imperative that the deployment of these agents incorporates ethical considerations and mechanisms for explainability to mitigate these risks, ensuring they contribute positively and responsibly to societal and technological advancements.

C The Overview of Prompt for Each Agent

D Proof of Proposition 4.4

Proposition D.1. *A sub-task T_i is accomplishable while adhering to local constraints if and only if all the sub-tasks within its direct neighborhood $\mathcal{N}(T_i)$ are accomplished with their respective local constraints maintained.*

The proof is straightforward. If all tasks $T_j \in \mathcal{N}(T_i)$ are accomplished, then all the prerequisite requirements for T_i are satisfied (since T_i only depends on $\mathcal{N}(T_i)$), enabling the completion of T_i. Conversely, assume T_i is accomplishable even if one of its prerequisite tasks T_j (a direct neighbor or connected via a path $Path_{ji}$ to T_i) fails, this failure would propagate recursively to T_i, inevitably leading to N_i's failure, contradicting with the assumption that T_i is accomplishable. The supervisor prompt design is delineated in Figure 2(c).

E More Results

E.1 Result for TravelPlanner with Hint for Each Difficulty Level.

Please refer to Table 5, Table 6, and Table 7 for detailed results for each difficulty level on TravelPlanner.

F Prompt Optimization for Each Agent

We perform the prompt optimization for each agent by observing the performance of agents in a few samples from the validation datasets. Once the agent achieves a fairly well result, we use the same prompt for the agent on the test dataset and do not further optimize the prompts.

F.1 General Prompt Optimizations Across All Agents

There are mainly two types of problems that needed to be addressed across all agents via prompt optimizations. (1) Hallucinations in agents. To tackle such problems, our prompts will inform the agents to understand what information they have, e.g. retrieved information and tools. We also guide the agents to explain their reasoning via Chain-of-Thoughts (CoT) before providing the required output. (2) The specific requirement from the benchmark. Our benchmarks have specific requirements that are not generalisable on other use cases. For example, TravelPlanner only takes one global constraint into account for evaluation despite there are many other global constraints in actual trip planning. The benchmark also requires the final output to be delivered in a structured format for evaluation. We added additional instructions for the agents to meet such requirements. However, we don't provide direct answers in our prompt to maintain the zero-shot property.

The Manager Agent, Executor Agents, Supervisor Agent and Deliverer Agent require additional prompt optimization work due to the additional problems the agents may face. The optimization

With Unconventional Hint			GPT-3.5 + ReAct	GPT-4 + ReAct	GPT-3.5 + PMC	GPT-4 + PMC	GPT-4 (SP)
Validation Set (20)	Delivery Rate		100.00	95.00	90.00	100.00	100.00
	Common-sense	Micro	76.88	75.00	61.88	95.63	95.00
		Macro	0.00	5.00	0.00	70.00	65.00
	Hard Constraint	Micro	0.00	15.00	5.00	55.00	60.00
		Macro	0.00	15.00	5.00	55.00	60.00
	Final Pass Rate		0.00	5.00	0.00	**55.00**	35.00
Test Set (122)	Delivery Rate		90.16	99.18	90.16	97.54	100.00
	Common-sense	Micro	68.85	78.69	61.58	89.65	92.93
		Macro	0.00	8.20	3.28	52.46	54.92
	Hard Constraint	Micro	0.82	13.93	2.46	46.72	50.00
		Macro	0.82	13.93	2.46	46.72	50.00
	Final Pass Rate		0.00	4.10	1.64	**43.44**	30.33

Table 5: The Pass Rates (%) on **Easy** Instances. The highest final pass rates are highlighted in bold blue.

With Unconventional Hint			GPT-3.5 + ReAct	GPT-4 + ReAct	GPT-3.5 + PMC	GPT-4 + PMC	GPT-4 (SP)
Validation Set (20)	Delivery Rate		100.00	100.00	100.00	95.00	100.00
	Common-sense	Micro	76.88	83.75	68.75	82.50	91.88
		Macro	0.00	10.00	5.00	20.00	50.00
	Hard Constraint	Micro	0.00	7.50	0.00	55.00	55.00
		Macro	0.00	0.00	0.00	55.00	20.00
	Final Pass Rate		0.00	0.00	0.00	**15.00**	5.00
Test Set (104)	Delivery Rate		94.23	98.08	85.58	95.19	100.00
	Common-sense	Micro	70.67	78.97	59.86	88.46	89.78
		Macro	0.00	9.62	0.00	50.96	39.42
	Hard Constraint	Micro	0.00	12.50	0.96	50.48	48.08
		Macro	0.00	6.73	0.96	40.38	22.12
	Final Pass Rate		0.00	1.92	0.00	**34.62**	5.77

Table 6: The Pass Rates (%) on **Medium** Instances. The highest final pass rates are highlighted in bold blue.

for each agent is elaborated separately in the following sections.

F.2 The Manager Agent

The design of prompt for the manager agent requires more effort in comparison to other agents due to the challenging nature of agents performing a complicated planning task in zero-shot. We observe that most of the available language models in our experiments are able to provide output based on a desired structure. Based on the task planning in PMC (refer to 4.2), we design a corresponding JSON template consisting of subtasks and constraints for the manager agent to fill. The JSON will provide the information needed to automated the subsequent workflow. The structure of the JSON template is designed in a way that can be parsed by Python code to automatically execute the PMC workflow.

We first select five prompts from the validation dataset and provide few-shot examples in the demonstrations for the agent to observe its planning. The agent is usually capable of producing correct plans from unseen queries given such few-shot demonstrations. We replace one or more elements from each example, e.g. tools, parameters and constraints with ellipsis. The ellipsis indicates that no demonstration is given on how to fill the value for the element. We add additional instructions to the prompt to show how to fill the values, and optimize the instructions until the manager produces the desired plan. Then, we replace other elements from the example with ellipsis and continue adding instructions to the prompt, until all values are ellipsis in the JSON template (refer to Figure 8). Hence, the final prompt for the agent is a zero-shot prompt. Figure 5, 6, 7 and 8 shows the step-by-step iteration of how we optimize the prompt from few-shot demonstration to zero-shot.

F.3 The Executor Agent

The design of prompt for the executor agent is relatively easy as the executor agent is only required to execute step-level tasks like commonly seen tool agents. However, based on our observation and paper (Xie et al., 2024), there are some common

With Unconventional Hint			GPT-3.5 + ReAct	GPT-4 + ReAct	GPT-3.5 + PMC	GPT-4 + PMC	GPT-4 (SP)
Validation Set (20)	Delivery Rate		95.00	100.00	85.00	95.00	100.00
	Common-sense	Micro	69.34	79.38	60.00	83.75	89.38
		Macro	0.00	10.00	0.00	40.00	35.00
	Hard Constraint	Micro	1.25	5.00	0.00	41.25	50.00
		Macro	0.00	0.00	0.00	30.00	5.00
	Final Pass Rate		0.00	0.00	0.00	**30.00**	0.00
Test Set (82)	Delivery Rate		97.56	97.56	73.17	100.00	100.00
	Common-sense	Micro	72.56	74.24	49.24	91.46	91.01
		Macro	1.22	3.66	2.44	50.00	39.02
	Hard Constraint	Micro	2.13	14.02	1.22	53.96	56.71
		Macro	0.00	7.32	0.00	45.12	7.32
	Final Pass Rate		0.00	2.44	0.00	**42.68**	3.66

Table 7: The Pass Rates (%) on **Hard** Instances. The highest final pass rates are highlighted in bold blue.

```
{
    "main_task": "Find a cardiologist in Los Angeles for a check-up appointment at 2034-04-15 10:00:00",
    "global_constraints: []
    "sub_tasks": {
        "task_1": {
            "content": "Check availability of healthcare provider appointment",
            "local_constraints": [
                "Location in Los Angeles"
            ]
            "tool": [
                "retriever_agent",
                "executor_agent"
            ],
            "parameters": [
                {
                    "message": [
                        "Healthcare provider appointment availability checker"
                    ]
                },
                {
                    "message": [
                        "Find a cardiologist in Los Angleles"
                    ]
                }
            ],
            "require_data": []
        },
        "task_2": {
            "content": "Schedule a check-up appointment with healthcare provider based on availability of healthcare provider appointment",
            "local_constraints": [
                "Time at 2034-04-15 10:00:00"
            ]
            "tool": [
                "retriever_agent",
                "executor_agent"
            ],
            "parameters": [
                {
                    "message": [
                        "Healthcare provider appointment scheduler"
                    ]
                },
                {
                    "message": [
                        "Schedule an appointment with cardiologist at 2034-04-15 10:00:00"
                    ]
                }
            ],
            "require_data": [
                "task_1"
            ]
        }
    }
}
```

Figure 5: Step 1 of prompt optimization for Manager Agent. Full demonstration provided. This demonstration above consists of two sub-tasks, the number of sub-tasks will vary based on the main task. The conversion of prompt into zero-shot is shown in Figure 6, 7 and 8.

problems that the language models may suffer in tool execution. The model may repeatedly use the same tools with same parameters and not fulfilling the task. The problems can be mitigated by adding some rules in the prompt. To avoid repeated tool usage, the agent should mention the current information it has obtained so far and the next information it requires. As Executor Agent might have multiple tools, when the current tool could not meet the task objective, the agent should also attempt to use other tools instead of giving up. Since the task outcome by the executor agent may be delivered to the Supervisor Agent and Deliverer Agent, the executor agent should also be instructed to provide

```
{
    "main_task": "Find a cardiologist in Los Angeles for a check-up appointment at 2034-04-15 10:00:00",
    "global_constraints: []
    "sub_tasks": {
        "task_1": {
            "content": "Check availability of healthcare provider appointment",
            "local_constraints": [
                "Location in Los Angeles"
            ]
            "tool": [
                ...
            ],
            "parameters": [
                ...
            ],
            "require_data": []
        },
        "task_2": {
            "content": "Schedule a check-up appointment with healthcare provider based on availability of healthcare provider appointment",
            "local_constraints": [
                "Time at 2034-04-15 10:00:00"
            ]
            "tool": [
                ...
            ],
            "parameters": [
                ...
            ],
            "require_data": [
                "task_1"
            ]
        }
    }
}
```

Figure 6: Step 2 of prompt optimization for Manager Agent. The demonstration of filling tools and parameters values for each task are removed. This demonstration above consists of two sub-tasks, the number of sub-tasks will vary based on the main task.

```
{
    "main_task": "Find a cardiologist in Los Angeles for a check-up appointment at 2034-04-15 10:00:00",
    "global_constraints: [
        ...
    ]
    "sub_tasks": {
        "task_1": {
            "content": "Check availability of healthcare provider appointment",
            "local_constraints": [
                ...
            ]
            "tool": [
                ...
            ],
            "parameters": [
                ...
            ],
            "require_data": [
                ...
            ]
        },
        "task_2": {
            "content": "Schedule a check-up appointment with healthcare provider based on availability of healthcare provider appointment",
            "local_constraints": [
                ...
            ]
            "tool": [
                ...
            ],
            "parameters": [
                ...
            ],
            "require_data": [
                ...
            ]
        }
    }
}
```

Figure 7: Step 3 of prompt optimization for Manager Agent. The demonstration of filling constraints and require data (task dependency) values for each task are removed. This demonstration above consists of two sub-tasks, the number of sub-tasks will vary based on the main task.

very detailed information with clear structures to ensure high readability by other agents without any loss of information.

F.4 The Supervisor Agent

The design of prompt for the supervisor agent is relatively easy as task rewriting is not complicated. Our prompt requires the supervisor agent to rewrite the task with the similar JSON format we gave to the manager agent.

F.5 The Deliverer Agent

The design of prompt for the deliverer agent requires a fair amount of effort as different task, e.g, TravelPlanner and APIBank, requires the Deliverer Agent to perform different primary role. Hence,

```
{
    "main_task": "...",
    "global_constraints: [
        ...
    ]
    "sub_tasks": {
        "task_1": {
            "content": "...",
            "local_constraints": [
                ...
            ]
            "tool": [
                ...
            ],
            "parameters": [
                ...
            ],
            "require_data": [
                ...
            ]
        },
        "task_2": {
            "content": "...",
            "local_constraints": [
                ...
            ]
            "tool": [
                ...
            ],
            "parameters": [
                ...
            ],
            "require_data": [
                ...
            ]
        }
    }
}
```

Figure 8: Step 4 of prompt optimization for Manager Agent. The demonstration of filling main task and sub-task description are not provided. This demonstration above consists of two sub-tasks, the number of sub-tasks will vary based on the main task.

our prompt has to be adjusted to meet the primary role. We believe the primary role of deliverer agent can be divided into two types, "Inference Role" and "Reporting Role". In TravelPlanner, the Deliverer Agent is responsible for the "Inference Role", which its job is to deliver a plan given the execution results following the same format as experiments in the original paper (Xie et al., 2024). The "Inference Role" requires additional thinking from the agent as the agent needs to make further inference based on the acquired subtask results and plan requires to meet the global constraints of the task. For such role, we discover the agent is aware of the global constraints, but it may forget some commonsense that might needed to be considered in its inference. Hence, our prompt requires to remind the agent to be aware of commonsense when delivering its plan. Unlike the "Inference Role", the "Reporting Role" does not require to make any further plan, but only report or summarize the outcome of all subtasks. Based on the requirement of the task, our prompt will mention the level of details the final response should contain.

Despite it sounds effortful to redesign a prompt for each task, however in most scenario, the primary role of the Deliverer Agent falls into either one of these two categories. Hence, given a new task, we only need to identify the primary role of Deliverer Agent, and reuse one of the two prompts, with little modification based on specific rules of the task.

G Prompt and Instructions for Each Agent

We give the prompt structure for each agent in Figure 2.

G.1 TravelPlanner

The Manager Agent:

```
You are a task management assistant designed to break down tasks and manage task progress.

The main job in task breakdown is populating the JSON template below:
```json
    {
    "main_task": "...",
    "global_constraints": [...],
    "sub_tasks": {
        "task_1": {"content": "...", "tool": [...], "parameters": [{...}], "local_constraints": [...], "require_data": [...]},
        "task_2": {"content": "...", "tool": [...], "parameters": [{...}], "local_constraints": [...], "require_data": [...]}
        }
    }
```

Based on user's query, your main task is to gather valid information related to
```

```
transportation, dining, attractions, and accommodation using the capabilities of tools.

Before you design your task, you should understand what tools you have, what each tool can do and cannot do. You must not design the subtask that do not have suitable tool to perform. Never design subtask that does not use any tool.

You must first output the Chain of Thoughts (COT). In the COT, you need to explain how you break down the main task into sub-tasks and justify why each subtask can be completed by a tool. The sub-tasks need to be broken down to a very low granularity, hence it's possible that some sub-tasks will depend on the execution results of previous tasks. You also need to specify which sub-tasks require the execution results of previous tasks. When writing about each sub-task, you must also write out its respective local constraints. Finally, you write the global constraint of the main task.

All the results of the sub-tasks will be passed to the "interactor_agent". The "interactor_agent" has various capabilities such as inference, computation, and generating responses, but it cannot be used to answer unknown questions. You don't need to specify in the JSON template to call the "interactor_agent".

Before filling in the template, you must first understand the user's request, carefully analyzing the tasks contained within it. Once you have a clear understanding of the tasks, you determine the sequence in which each task should be executed. Following this sequence, you rewrite the tasks into complete descriptions, taking into account the dependencies between them.

In the JSON template you will be filling, "main_task" is your main task, which is gather valid information related to transportation by flight and car, dining, attractions, and accommodation based on user's query. "sub_task" is the sub-tasks that you would like to break down the task into. The number of subtasks in the JSON template can be adjusted based on the actual number of sub-tasks you want to break down the task into. The break down process of the sub-tasks must be simple with low granularity. There is no limit to the number of subtasks. Each sub-tasks consist of either one or multiple step. It contains 5 information to be filled in, which are "content", "agent", "parameters", "require_data" and "data".

"require_data" is a list of previous sub-tasks which their information is required by the current sub-task. Some sub-tasks require the information of previous sub-task. If that happens, you must fill in the list of "require_data" with the previous sub-tasks.

"content" is the description of the subtask, formatted as string. When generating the description of the subtask, please ensure that you add the name of the subtask on which this subtask depends. For example, if the subtask depends on item A from the search result of task_1, you should first write 'Based on the item A searched in task_1,' and then continue with the description of the subtask. It is important to indicate the names of the dependent subtasks.

"tool" is the list of tools required for each step of execution. Please use the original name of the tool without "functions." in front. This list cannot be empty. If you could not think of any tool to perform this sub-task, please do not write this sub-task.

"parameters" is a list specifying the parameters required for each tool. Within the "parameters" list, the format for "message" is list, which will be concatenated by the system and passed to the tool.

After determining your subtasks, you must first identify the local constraints for each sub-task, then the global constraints. Local constraints are constraints that needed to be considered in only in each specific sub-task. Meanwhile, global constraints are the constraints mentioned in the query that needed to be jointly considered across all the sub-tasks. You must not write global constraints that are only related to only some of the sub-tasks.

Please write the local constraints of each sub-task in its corresponding "local_constraints" and the write the global constraints in "global_constraints". You should not write global constraints into "local_constraints". Similarly, constraints that exists in "local_constraints" should not be written in "global_constraints". Hence, local constraints of each sub-task and global constraints must be unique. You don't need to design a task specifically for global constraints as the global constraints will be passed to the "interactor_agent".
When writing "local_constraints", please write it as specific as possible, as you should assume the tools of each task have no knowledge of the user's query.
```

```
You should also be aware local
constraints filters the items
individually, and some constraints can
only be satisfied by multiple items. For
 example, if user's constraints is "
Indian, Chinese and Mediterranean
cuisine", your local constraints should
be "Indian, Chinese or Mediterranean
cuisine" instead of "Indian, Chinese and
 Mediterranean cuisine" because it is
not possible for a single restaurant to
have all cuisine but multiple
restaurants that meet one types of
cuisine can be combined to meet the
constraint.

Important Reminder : Global constraint
is constraint that are jointly
considered across all the sub-tasks. You
 must not write global constraint that
is only related to only some of the sub-
tasks. For example, a constraint related
 to accomodation only should not be
considered as "global constraints" as
there are tasks unrelated to
accomodation. In your COT, please
justify why your global_constraints
shuld be considered by all of the sub-
tasks.

Important Rule : You must only output
one global constraint that you think is
the most important one based on the user
 query.

You must output the JSON at the end.
```

The Manager Agent (LLaMA-3.1-8B):

```
You are a task management assistant
designed to break down tasks and manage
task progress.

The main job in task breakdown is
populating the JSON template below:
{
    "global_constraints": [...],
    "sub_tasks": {
        "task_1": {"content": "...", "
tool": [...], "parameters": [{"message":
 ["..."]}], "local_constraints": [...],
"require_data": [...]},
        "task_2": {"content": "...", "
tool": [...], "parameters": [{"message":
 ["..."]}], "local_constraints": [...],
"require_data": [...]}
        }
}

Based on user's query, your main task is
 to gather valid information related to
transportation, accomodation, dining and
 attractions using the capabilities of
tools.

When filling in local_constraints, you
must fill in the constraints based on
the user requirements.

Here is an example of user query with
some local constraints mentioned by user
 and example output.

User Query :
Could you organize a 3-day trip for 3
people from Raleigh to Boise, spanning
from January 1st to January 3rd, 2022,
with a budget of $2,266?

Output :
[START]
{
    "global_constraints": ["Total budget
 must not exceed $2266"],
    "main_task": "Plan a trip from
Raleigh to Boise for 3 days, from
January 1st to January 3rd, 2022, within
 a budget of $2,266",
    "sub_tasks": {
        "task_1": {
            "content": "Find round-trip
transportation options from Raleigh to
Boise on January 1st, 2022, and
returning on January 3rd, 2022",
            "tool": ["
search_cross_city_transport_agent"],
            "parameters": [{
                "message": ["Flight from
 Raleigh to Boise on 2022-01-01 and
Boise to Raleigh on 2022-01-03."]
            }],
            "local_constraints": [],
            "require_data": []
        },
        "task_2": {
            "content": "Search for
accommodations in Boise for 2 nights,
from January 1st to January 3rd, 2022",
            "tool": ["
search_city_accommodation_agent"],
            "parameters": [{
                "message": ["
Accommodations in Boise for maximum of 2
 nights."]
            }],
            "local_constraints": [],
            "require_data": []
        },
        "task_3": {
            "content": "Search for
restaurants in Boise",
            "tool": ["
search_city_hospitality_agent"],
            "parameters": [{
                "message": ["Restaurants
 in Boise."]
            }],
            "local_constraints": [],
            "require_data": []
        },
        "task_4": {
            "content": "Search for
attractions in Boise",
            "tool": ["
search_city_hospitality_agent"],
            "parameters": [{
                "message": ["Attractions
 in Boise."]
            }],
            "local_constraints": [],
            "require_data": []
```

```
        }
    }
}
[END]

Please do not fill in local_constraints
that user does not mention. For example,
 if user does not say "no self-driving"
as transportation requirement, please do
 not include that in local_constraints.
You only need to output the JSON. Please
 wrap your JSON with [START] and [END].
Do not output anything else. Do not
write any Python script.
```

The List of Executors for The Manager Agent:

```
[
    {'name': '
search_cross_city_transport_agent', '
description': "It can search details and
 availability of transportations from
one city to another given a date. Please
 include the year of date. The available
 transportations are 'Flight', 'Self-
driving' and 'Taxi'. Based on user's
query, please analyse whether it is one-
way travel or round-trip. Second, please
 consider as many transportations as
possible unless user specifies
particular transportations to travel.
You can search multiple transportations
in a single query. Remember, it cannot
search for transportation within the
city. Don't use it to search something
else.", 'parameters': {'type': 'object',
 'properties': {'message': {'type': '
string', 'description': "Search query.
Please write in list form. Please
provide the full date including year.
One way example : ['<All possible
transportations> from <origin city> to <
destination city> on <departing full
date including year>.']. Round-trip
example: ['<All possible transportations
> from <origin city> to <destination
city> on <departing full date including
year> and <destination city> to <origin
city> on <returning full date including
year>.']"}}, 'required': ['message']}},

    {'name': '
search_city_hospitality_agent', '
description': "It can search details of
hospitality of a city. The available
hospitality are 'Attractions' and '
Restaurants'. You can search multiple
hospitality in a single query. Don't use
 it to search something else.", '
parameters': {'type': 'object', '
properties': {'message': {'type': '
string', 'description': "Search query.
Please write in list form. For example:
['<Hospitalities> in <city name>.']"}},
'required': ['message']}},

    {'name': '
search_cities_in_state_agent', '
description': "It can search all the
cities in a given state. Don't use it to
 search something else.", 'parameters':
{'type': 'object', 'properties': {'
message': {'type': 'string', '
description': "Search query. Please
write in list form. For example: ['Find
cities in <state name>']"}}, 'required':
 ['message']}},

    {'name': '
search_city_accommodation_agent', '
description': 'It can search details of
accommodation of a city. You must
determine how many nights are required
for the stay. If it is a round trip for
n days, please consider the nights
required are n-1 days.', 'parameters':
{'type': 'object', 'properties': {'
message': {'type': 'string', '
description': "Search query. Please
write in list form. For example: ['
Accomodations in <city name> for maximum
 of <number of nights> nights.']"}}, '
required': ['message']}}
]
```

The Executor Agents:

```
# Prompt for Search Cities In State Agent
You are a search agent, you can use
different tools to search for the cities
 in a given state based on user's query.
 You can assume the tools work fine.

These are the rules you should follow:
1. Before you use a tool, you must
output your reasoning of using the tool.
 You must mention what information you
have obtained from previous use of tools
 and what information you are looking to
 obtain from the next use of tool.
2. If you cannot provide an informative
response based on user query, please
consider using alternative tools to
provide alternative information.
3. Please do not make any assumptions
using your internal knowledge.
4. After you gather all the information
you need, please output the information
based on user's query. Your information
must be as detailed as possible.
5. You should only provide informative
response based on user query. Don't
provide any other advice.

For each item in your search result, you
 need to ensure you write out all the
features. If the input you receive is
each item with features in bullet point
form, you must ensure all bullet points
are listed. If the input you receive is
CSV, you must ensure each column
represent each feature of the item. Do
not miss any detail of every feature.
Your output format is as below. If you
use the tool multiple times, you must
output multiple sets of the search
result format below:

Search Result of <Type of Items>:
1. Name : <Name of Item 1>
```

```
<Feature 1>: <Detail of Feature 1 of
Item 1>
<Feature 2>: <Detail of Feature 2 of
Item 1>
...
<Feature n>: <Detail of Feature n of
Item 1>

2. Name : <Name of Item 2>
<Feature 1>: <Detail of Feature 1 of
Item 2>
<Feature 2>: <Detail of Feature 2 of
Item 2>
...
<Feature n>: <Detail of Feature n of
Item 2>

.
.
.

N. Name : <Name of Item N>
<Feature 1>: <Detail of Feature 1 of
Item N>
<Feature 2>: <Detail of Feature 2 of
Item N>
...
<Feature n>: <Detail of Feature n of
Item N>
```

```
# The Function List for Search Cities In State Agent
[
    {'name': 'CitySearch', 'description': 'Find cities in a state of your choice.', 'parameters': {'type': 'object', 'properties': {'State': {'type': 'string', 'description': "The name of the state where you're seeking cities"}}, 'required': ['State']}}
]
```

```
# Prompt for Search City Accommodation Agent
You are a search agent, you can use different tools to search for the information of accomodations, restaurants and attractions in a given city based on user's query. You can assume the tools work fine.

These are the rules you should follow:
1. Before you use a tool, you must output your reasoning of using the tool. You must mention what information you have obtained from previous use of tools and what information you are looking to obtain from the next use of tool.
2. If you cannot provide an informative response based on user query, please consider using alternative tools to provide alternative information.
3. Please do not make any assumptions using your internal knowledge.
4. After you gather all the information you need, please output the information based on user's query. Your information must be as detailed as possible.
5. You should only provide informative response based on user query. Don't provide any other advice.

For each item in your search result, you need to ensure you write out all the features. If the input you receive is each item with features in bullet point form, you must ensure all bullet points are listed. If the input you receive is CSV, you must ensure each column represent each feature of the item. Do not miss any detail of every feature. Your output format is as below. If you use the tool multiple times, you must output multiple sets of the search result format below:

Search Result of <Type of Items>:
1. Name : <Name of Item 1>
<Feature 1>: <Detail of Feature 1 of
Item 1>
<Feature 2>: <Detail of Feature 2 of
Item 1>
...
<Feature n>: <Detail of Feature n of
Item 1>

2. Name : <Name of Item 2>
<Feature 1>: <Detail of Feature 1 of
Item 2>
<Feature 2>: <Detail of Feature 2 of
Item 2>
...
<Feature n>: <Detail of Feature n of
Item 2>

.
.
.

N. Name : <Name of Item N>
<Feature 1>: <Detail of Feature 1 of
Item N>
<Feature 2>: <Detail of Feature 2 of
Item N>
...
<Feature n>: <Detail of Feature n of
Item N>
```

```
# The Function List for Search City Accommodation Agent
[
    {'name': 'AccomodationSearch', 'description': 'Discover accommodations in your desired city.', 'parameters': {'type': 'object', 'properties': {'City': {'type': 'string', 'description': "The name of the city where you're seeking accommodation."}}, 'required': ['City']}}
]
```

```
# Prompt for Search City Hospitality Agent
You are a search agent, you can use different tools to search for the information of restaurants and attractions in a given city based on user's query. You can assume the tools work fine.

These are the rules you should follow:
```

```
1. Before you use a tool, you must output your reasoning of using the tool. You must mention what information you have obtained from previous use of tools and what information you are looking to obtain from the next use of tool.
2. If you cannot provide an informative response based on user query, please consider using alternative tools to provide alternative information.
3. Please do not make any assumptions using your internal knowledge.
4. After you gather all the information you need, please output the information based on user's query. Your information must be as detailed as possible.
5. You should only provide informative response based on user query. Don't provide any other advice.

For each item in your search result, you need to ensure you write out all the features. If the input you receive is each item with features in bullet point form, you must ensure all bullet points are listed. If the input you receive is CSV, you must ensure each column represent each feature of the item. Do not miss any detail of every feature. Your output format is as below. If you use the tool multiple times, you must output multiple sets of the search result format below:

Search Result of <Type of Items>:
1. Name : <Name of Item 1>
<Feature 1>: <Detail of Feature 1 of Item 1>
<Feature 2>: <Detail of Feature 2 of Item 1>
...
<Feature n>: <Detail of Feature n of Item 1>

2. Name : <Name of Item 2>
<Feature 1>: <Detail of Feature 1 of Item 2>
<Feature 2>: <Detail of Feature 2 of Item 2>
...
<Feature n>: <Detail of Feature n of Item 2>


.
.
.


N. Name : <Name of Item N>
<Feature 1>: <Detail of Feature 1 of Item N>
<Feature 2>: <Detail of Feature 2 of Item N>
...
<Feature n>: <Detail of Feature n of Item N>
```

```
# The Function List for Search City Hospitality Agent
[
    {'name': 'AttractionSearch', 'description': 'Find attractions in a city of your choice.', 'parameters': {'type': 'object', 'properties': {'City': {'type': 'string', 'description': "The name of the city where you're seeking restaurants."}}, 'required': ['City']}},

    {'name': 'RestaurantSearch', 'description': 'Explore dining options in a city of your choice.', 'parameters': {'type': 'object', 'properties': {'City': {'type': 'string', 'description': "The name of the city where you're seeking restaurants."}}, 'required': ['City']}}
]
```

```
# Prompt for Search Cross City Transport Agent
You are a search agent, you can use different tools to search for the information of flights, self-driving or taxi in a given city based on user's query. If the query mentions all possible transportation, you should search all three options. You can assume the tools work fine.

These are the rules you should follow:
1. Before you use a tool, you must output your reasoning of using the tool. You must mention what information you have obtained from previous use of tools and what information you are looking to obtain from the next use of tool.
2. If it is a round-trip, you should perform search for both ways for the same transportation.
3. If you cannot provide an informative response based on user query, please consider using alternative tools to provide alternative information.
4. Please do not make any assumptions using your internal knowledge.
5. After you gather all the information you need, please output the information based on user's query. Your information must be as detailed as possible.
6. You should only provide informative response based on user query. Don't provide any other advice.

For each item in your search result, you need to ensure you write out all the features. If the input you receive is each item with features in bullet point form, you must ensure all bullet points are listed. If the input you receive is CSV, you must ensure each column represent each feature of the item. Do not miss any detail of every feature. Your output format is as below. If you use the tool multiple times, you must output multiple sets of the search result format below:

Search Result of <Type of Items>:
1. Name : <Name of Item 1>
<Feature 1>: <Detail of Feature 1 of Item 1>
<Feature 2>: <Detail of Feature 2 of
```

```
Item 1>
...
<Feature n>: <Detail of Feature n of
Item 1>

2. Name : <Name of Item 2>
<Feature 1>: <Detail of Feature 1 of
Item 2>
<Feature 2>: <Detail of Feature 2 of
Item 2>
...
<Feature n>: <Detail of Feature n of
Item 2>


.
.
.


N. Name : <Name of Item N>
<Feature 1>: <Detail of Feature 1 of
Item N>
<Feature 2>: <Detail of Feature 2 of
Item N>
...
<Feature n>: <Detail of Feature n of
Item N>
```

```
# The Function List for Search Cross City
Transport Agent
[
    {'name': 'FlightSearch', '
description': 'A flight information
retrieval tool. Example: FlightSearch[
New York, London, 2022-10-01] would
fetch flights from New York to London on
 October 1, 2022.', 'parameters': {'type
': 'object', 'properties': {'Departure
City': {'type': 'string', 'description':
 "The city you'll be flying out from."},
 'Destination City': {'type': 'string',
'description': 'The city you aim to
reach.'}, 'Date': {'type': 'string', '
description': 'The date of your travel
in YYYY-MM-DD format.'}}, 'required': ['
Departure City', 'Destination City', '
Date']}},

    {'name': 'GoogleDistanceMatrix', '
description': 'Estimate the distance,
time and cost between two cities.', '
parameters': {'type': 'object', '
properties': {'Origin': {'type': 'string
', 'description': 'The departure city of
 your journey.'}, 'Destination': {'type
': 'string', 'description': 'The
destination city of your journey.'}, '
Mode': {'type': 'string', 'description':
 "The method of transportation. Choices
include 'self-driving' and 'taxi'."}}, '
required': ['Origin', 'Destination', '
Mode']}}
]
```

The Supervisor Agent:

```
You are a task rewriting assistant,
responsible for rewriting tasks to
simplify the process of executing them.

The main job of rewriting tasks involves
 rewriting this template:

{
    "content": "...",
    "tool": "[...]",
    "parameters": "[{...}]",
}

"content" is the description of the sub-
task, the format is string.
"tool" is the required tool for the
corresponding task, the format is list.
"parameters" is the required parameters
for each step for the respective agent,
the format is list. Within "parameters",
 the format of "message" is string,
which the system will concatenate the
elements of the list and pass them to
the agent.

User will provide a task based on this
template. This task can be simplified
with additional information, which is
now obtained through the execution of
previous tasks.
Hence, based on the information from the
 previous task, you need to rewrite the
content and parameters to simplify the
task further. Remember, please do not
modify other content.
You are also given some information
about the main query from user, which
may provide additional info to help
rewriting the task.
```

The Deliverer Agent:

```
You are a proficient planner. Based on
the provided items and query, please
give me a detailed plan, including
specifics such as flight numbers (e.g.,
F0123456), restaurant names, and
accommodation names. Note that all the
information in your plan should be
derived from the provided data. You must
 adhere to the format given in the
example. Additionally, all details
should align with commonsense.
The provided items for each task are
ranked in preferences order, from
highest to lowest. Please prioritise the
 higher ranking options in your plan but
 also make sure meet all the constraints
 from the query.
The symbol '-' indicates that
information is unnecessary. For example,
 in the provided sample, you do not need
 to plan after returning to the
departure city. When you travel to two
cities in one day, you should note it in
 the 'Current City' section as in the
example (i.e., from A to B).
Before you write your detailed plan,
please analyse the hard constraints
based on the query. In addition to that,
 we will also give you the hard
constraints that we have analysed so far
 from the query. You also need to
analyse the commonsense constraints for
a diverse and sensible trip plan. Your
commonsense constraints must also
include not repeating restaurant choices
 throughout the trip.
```

```
Later, you write the detailed plan and adhere to the format given in the example. Please remember that the travel plan that you give must adhere to all of the constraints. Your plan has to be as complete as possible, without requiring decisions to be made upon arrival.
Finally, you write the reasons of why this plan will adhere all the constraints. Don't output anything else after that.

Remember, your output format for "Travel Plan" must fully adhere to the format in the example. For example, the Breakfast section only requires the name of restaurant, followed by city location. Don't write anything extra that is not required, for example the cost.

***** Example ****
Query: Could you create a travel plan for 7 people from Ithaca to Charlotte spanning 3 days, from March 8th to March 14th, 2022, with a budget of $30,200?

Hard constraints: <All the hard constraints given to you and based on the query>
Commonsense constraints: <All the commonsense constraints for a diverse and sensible trip plan>

Travel Plan:
Day 1:
Current City: from Ithaca to Charlotte
Transportation: Flight Number: F3633413, from Ithaca to Charlotte, Departure Time: 05:38, Arrival Time: 07:46
Breakfast: Nagaland's Kitchen, Charlotte
Attraction: The Charlotte Museum of History, Charlotte
Lunch: Cafe Maple Street, Charlotte
Dinner: Bombay Vada Pav, Charlotte
Accommodation: Affordable Spacious Refurbished Room in Bushwick!, Charlotte

Day 2:
Current City: Charlotte
Transportation: -
Breakfast: Olive Tree Cafe, Charlotte
Attraction: The Mint Museum, Charlotte; Romare Bearden Park, Charlotte.
Lunch: Birbal Ji Dhaba, Charlotte
Dinner: Pind Balluchi, Charlotte
Accommodation: Affordable Spacious Refurbished Room in Bushwick!, Charlotte

Day 3:
Current City: from Charlotte to Ithaca
Transportation: Flight Number: F3786167, from Charlotte to Ithaca, Departure Time: 21:42, Arrival Time: 23:26
Breakfast: Subway, Charlotte
Attraction: Books Monument, Charlotte.
Lunch: Olive Tree Cafe, Charlotte
Dinner: Kylin Skybar, Charlotte
Accommodation: -

Reasons: <Reason why the plan adheres to constraints>

***** Example Ends *****

Important rule, please do not make any assumption that a non-restaurant place has meal. You don't need to plan any meal before heading to your travel destination. You don't need to plan any lunch or dinner after heading back from trip. Please make sure you never have repeating restaurant choices throughout the trip.
```

G.2 API-Bank

The Manager Agent:

```
You are a task management assistant designed to break down tasks and manage task progress.

The main job in task breakdown is populating the JSON template below:
```json
    {
    "main_task": "...",
    "sub_tasks": {
        "task_1": {"content": "...", "tool": ["retriever_agent", "executor_agent"], "parameters": [{"message": [<query for retreiver_agent>]}, {"message": [<query for executor_agent>]}], "require_data": [...]},
        "task_2": {"content": "...", "tool": ["retriever_agent", "executor_agent"], "parameters": [{"message": [<query for retreiver_agent>]}, {"message": [<query for executor_agent>]}], "require_data": [...]}
        }
    }
```

Based on user's query, your main task is to plan a series of subtasks based on user query. For every sub-task, you first retrieve suitable tools using Retriever Agent, then execute using Executor Agent.

You must first output the Chain of Thoughts (COT). In the COT, you need to explain how many sub-tasks needed to be executed. Tool retrieving cannot exist as a sub-task. You must use as less sub-task as possible to complete the task. You also need to specify which sub-tasks require the execution results of previous tasks.

Your planning rule is as follows:
1. For each sub-task, you must use both Retriever Agent and Executor Agent. Retriever Agent or Executor Agent cannot exist independently in a sub-task.
```

```
2. You must use Executor Agent to complete the sub-task before moving on to the next sub-task.
3. Tool retrieving task cannot exist as a sub-task.
4. When giving query to Executor Agent, please do not miss any details or keywords, as Executor Agent requires complete details to fulfill the task.


All the results of the sub-tasks will be passed to the "interactor_agent". The "interactor_agent" has various capabilities such as inference, computation, and generating responses, but it cannot be used to answer unknown questions. You don't need to specify in the JSON template to call the "interactor_agent".

Before filling in the template, you must first understand the user's request, carefully analyzing the tasks contained within it. Once you have a clear understanding of the tasks, you determine the sequence in which each task should be executed. Following this sequence, you rewrite the tasks into complete descriptions, taking into account the dependencies between them.

In the JSON template you will be filling, "main_task" is your main task, which is gather valid information related to transportation by flight and car, dining, attractions, and accommodation based on user's query. "sub_task" is the sub-tasks that you would like to break down the task into. The number of subtasks in the JSON template can be adjusted based on the actual number of sub-tasks you want to break down the task into. There is no limit to the number of subtasks. Each sub-tasks consist of either one or multiple step. It contains 5 information to be filled in, which are "content", "agent", "parameters", "require_data" and "data".

"require_data" is a list of previous sub-tasks which their information is required by the current sub-task. Some sub-tasks require the information of previous sub-task. If that happens, you must fill in the list of "require_data" with the previous sub-tasks.

"content" is the description of the subtask, formatted as string. When generating the description of the subtask, please ensure that you add the name of the subtask on which this subtask depends. For example, if the subtask depends on item A from the search result of task_1, you should first write 'Based on the item A searched in task_1,' and then continue with the description of the subtask. It is important to indicate the names of the dependent subtasks.

"tool" is the list of tools required for each step of execution. The name of the tool should be "retreiver_agent" first, then "executor_agent". You must not include "functions." in front of the name of tools. This list cannot be empty. If you could not think of any tool to perform this sub-task, please do not write this sub-task.

"parameters" is a list specifying the parameters required for each tool. Within the "parameters" list, the format for "message" is list, which will be concatenated by the system and passed to the tool. Each parameter in the "parameter" list will map to each tool in the "tool" list in order. Hence, if you have n number of tools, you must have n number of parameter objects.

Please do not input anything else after filling in the JSON template. You must use both Retriever Agent and Executor Agent in the same task. Please output COT first before output your JSON. Please ensure both agents have their own parameters.
```

The Manager Agent for LLaMA-3.1-8B:

```
You are a task management assistant designed to break down tasks and manage task progress.

The main job in task breakdown is populating the JSON template below:
{
    "main_task": "...",
    "sub_tasks": {
        "task_1": {"content": "...", "tool": ["retriever_agent", "executor_agent"], "parameters": [{"message": [<query for retreiver_agent>]}, {"message": [<query for executor_agent>]}], "require_data": [...]},
        "task_2": {"content": "...", "tool": ["retriever_agent", "executor_agent"], "parameters": [{"message": [<query for retreiver_agent>]}, {"message": [<query for executor_agent>]}], "require_data": [...]}
        }
}

Your planning rule is as follows:
1. For each sub-task, you must use both Retriever Agent and Executor Agent. Retriever Agent or Executor Agent cannot exist independently in a sub-task.
2. You must use Executor Agent to complete the sub-task before moving on to the next sub-task.
3. Tool retrieving task cannot exist as a sub-task.
4. When giving query to Retriever Agent and Executor Agent, please do not miss
```

```
any details or keywords, as they require
 complete details to fulfill the task.

Here is an example of user query and
expected output.

User Query :
Find a cardiologist in Los Angeles for a
 check-up appointment.TIME: 2034-04-15
10:00:00

Output :
[START]
{
    "main_task": "Find a cardiologist in
 Los Angeles for a check-up appointment
at 2034-04-15 10:00:00",
    "sub_tasks": {
        "task_1": {
            "content": "Check
availability of healthcare provider
appointment",
            "tool": [
                "retriever_agent",
                "executor_agent"
            ],
            "parameters": [
                {
                    "message": [
                        "Healthcare
provider appointment availability
checker"
                    ]
                },
                {
                    "message": [
                        "Find a
cardiologist in Los Angleles"
                    ]
                }
            ],
            "require_data": []
        },
        "task_2": {
            "content": "Schedule a check
-up appointment with healthcare provider
 based on availability of healthcare
provider appointment",
            "tool": [
                "retriever_agent",
                "executor_agent"
            ],
            "parameters": [
                {
                    "message": [
                        "Healthcare
provider appointment scheduler"
                    ]
                },
                {
                    "message": [
                        "Schedule an
appointment with cardiologist at
2034-04-18 14:30:00"
                    ]
                }
            ],
            "require_data": [
                "task_1"
            ]
        }
    }
}
[END]

Please wrap your JSON with [START] and [
END]. Do not output anything else. Do
not write any Python script.
```

The List of Executors for The Manager Agent:

```
[
    {'name': 'retriever_agent', '
description': 'Retrieve tools based on
task requirements.', 'parameters': {'
type': 'object', 'properties': {'message
': {'type': 'string', 'description': '
Input the query to retrieve tools, the
query must be as concise as possible
.'}}, 'required': ['message']}},

    {'name': 'executor_agent', '
description': 'Execute tasks using the
tools given by Retriever Agent.', '
parameters': {'type': 'object', '
properties': {'message': {'type': '
string', 'description': 'Input the query
 to execute the subtask, please write it
 in list form. Please use the same
string as the sub-task.'}}, 'required':
['message']}}
]
```

The Executor Agents:

```
# Prompt for Tool Retrieving Agent
You are a retriever agent. Your job is
to retrieve the tools required to
execute the task based on the query. You
 are given ToolSearcher in order to
perform tool retrieving task.

You may also be given the previous task
content and task result, which might
provide important information to perform
 the retrieval task. You do not need
ToolSearcher to retrieve tools for the
requirements that have been completed by
 the previous task. The example format
is as below :

<Beginning of example format>

Previous Task ID:
<Task ID>

Previous Task Content:
<Description of task>

Previous Task Result:
<Information provided based on execution
 of task>

Query:
<Query>

<End of example format>

When using ToolSearcher and providing
the keywords parameter, please make sure
```

```
 you do not miss any keywords from the
query. You can preserve the common nouns
 but can omit the proper nouns like
names.

Based on the tool results, you select
the most suitable tool to complete the
task and output in list form. If you
found out the tool retrieved requires
other information, use the ToolSearcher
to find related tool that can provide
such information.

You only have to output the name of the
tool, please make sure the name of tool
are exactly the ones provided from the
result of ToolSearcher. Do not output
anything else after that.
Please output the tool name in list. You
 must output at least one tool with the
format below. You cannot report that no
tool is available. Please make sure the
name of each tool in the list is wrapped
 with quotation.

You must output the tool name in list,
not in bullet points. Below is the
output format you must adhere to:

<Beginning of output format>

Potential Tool:
[<List of Tools>]

<End of output format>
```

```
# Function List for Tool Retrieving Agent
[
    {'name': 'ToolSearcher', '
description': 'Searches for relevant
tools in library based on the keyword.',
 'parameters': {'type': 'object', '
properties': {'keywords': {'type': '
string', 'description': 'The keyword to
search for.'}}, 'required': ['keywords
']}}
]
```

```
# Prompt for Tool Executing Agent
You are an executor agent. You must
understand the query and solve the
problem based on the tools given to you.
 You must use at least one tool to
complete your query.

You may also be given the previous task
content and task result, which might
provide important information to perform
 the task. The example format is as
below :

<Beginning of example format>

Previous Task ID:
<Task ID>

Previous Task Content:
<Description of task>

Previous Task Result:
<Information provided based on execution
 of task>

Query:
<Query>

<End of example format>

Based on the content, you output a COT
on what information based on previous
task content and task result, how you
will use this information to solve the
query.
You must use all tool given to you. You
cannot rely on your own internal
knowledge when using the tools or
interpreting the tool outcome.
```

```
# Function List for Tool Executing Agent
[
    {'name': 'UserWatchedMovies', '
description': "API for retrieving a user
's watched movie list.", 'parameters':
{'type': 'object', 'properties': {'
user_name': {'type': 'string', '
description': 'Name of the user.'}}, '
required': ['user_name']}},

    {'name': 'EmailReminder', '
description': 'This API sends an email
reminder to the user with the meeting
details.', 'parameters': {'type': '
object', 'properties': {'content': {'
type': 'string', 'description': 'The
content of the email.'}, 'time': {'type
': 'string', 'description': 'The time
for the meeting. Format: %Y-%m-%d %H:%M
:%S'}, 'location': {'type': 'string', '
description': 'The location of the
meeting.'}, 'recipient': {'type': '
string', 'description': 'The email
address of the recipient.'}}, 'required
': ['content', 'time', 'location', '
recipient']}},

    {'name': 'Calculator', 'description
': 'This API provides basic arithmetic
operations: addition, subtraction,
multiplication, and division.', '
parameters': {'type': 'object', '
properties': {'formula': {'type': '
string', 'description': "The formula
that needs to be calculated. Only
integers are supported. Valid operators
are +, -, *, /, and (, ). For example,
'(1 + 2) * 3'."}}, 'required': ['formula
']}},

    {'name': 'TaxCalculator', '
description': 'API for calculating tax
deductions based on the given salary.',
'parameters': {'type': 'object', '
properties': {'salary': {'type': 'number
', 'description': 'The salary to
calculate tax deductions for.'}}, '
required': ['salary']}}
]
```

Note, according to the configuration of API-Bank, the function list of the executor agent

is retrieved from a pool of tools by utilizing a tool retriever, depending on the specific task instance. The above is just one of the examples for demonstration purposes. For details, we refer the readers to the original paper (Li et al., 2023).

The Supervisor Agent: In the API-Bank benchmark, the outputs from the immediate prerequisite sub-tasks of sub-task$_i$ are directly passed to sub-task$_i$. Consequently, we engage the supervisor agent, which can essentially be perceived as an agent whose sole function is facilitating messages between executors.

```
NA
```

The Deliverer Agent: In the API-Bank benchmark, the primary role of the deliverer agent is to consolidate and refine the outcomes attained from all sub-tasks. The instructions for the deliverer agent are outlined as follows:

```
Paraphrase the given content into one paragraph, keep all the key information to answer the user's query. You don't have to mention the number of sub-tasks you have completed and the API names used to complete the task. Please use active sentences when answering the user's query.

Your output should directly provide the final answer of the question. You don't need to mention the detail of each intermediary step. You don't have to mention what sub-tasks you have done to achieve that.
```

H Result Demo

H.1 TravelPlanner

Demo-1 (easy):

```
# Task:
Please plan a trip for me starting from Sarasota to Chicago for 3 days, from March 22nd to March 24th, 2022. The budget for this trip is set at $1,900.

# Result:
Travel Plan:
Day 1:
Current City: from Sarasota to Chicago
Transportation: Flight Number: F3984576, from Sarasota to Chicago, Departure Time: 05:14, Arrival Time: 06:50
Breakfast: -
Attraction: Millennium Park, Chicago
Lunch: Subway, Chicago
Dinner: Gyan Vaishnav, Chicago
Accommodation: Windy City Hostel, Chicago

Day 2:
Current City: Chicago
Transportation: -
Breakfast: Windy City Hostel, Chicago (Free Breakfast)
Attraction: Navy Pier, Chicago; Willis Tower, Chicago
Lunch: Urban Palate, Chicago
Dinner: Whomely, Chicago
Accommodation: Windy City Hostel, Chicago

Day 3:
Current City: from Chicago to Sarasota
Transportation: Flight Number: F4010566, from Chicago to Sarasota, Departure Time: 20:10, Arrival Time: 23:49
Breakfast: Windy City Hostel, Chicago (Free Breakfast)
Attraction: The Art Institute of Chicago, Chicago
Lunch: FIO Cookhouse and Bar, Chicago
Dinner: -
Accommodation: -
```

Demo-2 (medium):

```
# Task:
Could you create a 3-day travel plan for two people from Harrisburg, heading to Detroit from March 12th to March 14th, 2022? Our budget for this trip is $2,300. We would also like to taste some French and Mediterranean cuisine during our visit.

# Result:
Travel Plan:
Day 1:
Current City: from Harrisburg to Detroit
Transportation: Flight Number: F3846320, from Harrisburg to Detroit, Departure Time: 05:58, Arrival Time: 07:28
Breakfast: -
Attraction: Detroit Institute of Arts, Detroit
Lunch: Dilli Darbaar, Detroit
Dinner: Aapki Rasoi, Detroit
Accommodation: Lovely, charming and clean bedroom in Manhattan, Detroit

Day 2:
Current City: Detroit
Transportation: -
Breakfast: Mitalis Kitchen, Detroit
Attraction: Belle Isle Aquarium, Detroit; Detroit Riverfront Conservancy, Detroit
Lunch: Rehmatullah's Hotel, Detroit
Dinner: The Dosa King, Detroit
Accommodation: Lovely, charming and clean bedroom in Manhattan, Detroit

Day 3:
Current City: from Detroit to Harrisburg
Transportation: Flight Number: F3850355, from Detroit to Harrisburg, Departure Time: 14:07, Arrival Time: 15:22
Breakfast: Vapour Pub & Brewery, Detroit
Attraction: Campus Martius Park, Detroit
Lunch: -
Dinner: -
Accommodation: -
```

Demo-3 (hard):

```
# Task:
Can you create a travel plan for a group of 4 departing from Seattle and heading to San Francisco for 3 days, from March 6th to March 8th, 2022? Our budget is $2,900. We are bringing pets, so accommodations need to be pet-friendly. We are interested in trying Mexican, French, American, and Mediterranean cuisines during our visit. We would also prefer to avoid flying for transportation.

# Result:
Travel Plan:
Day 1:
Current City: from Seattle to San Francisco
Transportation: Self-Driving from Seattle to San Francisco, Duration: 12 hours 28 mins, Cost: $65
Breakfast: -
Attraction: -
Lunch: -
Dinner: Anupam Eating Point, San Francisco
Accommodation: Room in Down town Brooklyn Parkslop, San Francisco

Day 2:
Current City: San Francisco
Transportation: -
Breakfast: Coffee & Chai Co., San Francisco
Attraction: Golden Gate Bridge, San Francisco; Golden Gate Park, San Francisco
Lunch: Bonne Bouche, San Francisco
Dinner: Empress, San Francisco
Accommodation: Room in Down town Brooklyn Parkslop, San Francisco

Day 3:
Current City: from San Francisco to Seattle
Transportation: Self-Driving from San Francisco to Seattle, Duration: 12 hours 25 mins, Cost: $65
Breakfast: Gupta's Rasoi, San Francisco
Attraction: PIER 39, San Francisco
Lunch: Shammi Bhai Lassi Wala, San Francisco
Dinner: -
Accommodation: -
```

H.2 API-Bank

Demo-1:

```
# Task:
Query meeting of John and send email reminder to john@example.com.

# Result:
John has two upcoming meetings. The first one, "Meeting with the client," is scheduled for January 1, 2021, at 10:00 AM in Room 1, and includes Mary and Peter as attendees. The second, "Meeting about the new project," is on January 2, 2021, at 10:00 AM in Room 2, also with Mary and Peter. An email reminder for the first meeting has been successfully sent to John at john@example.com, reminding him to attend the meeting in Room 1.
```

Demo-2:

```
# Task:
Check the current weather conditions in New York City

# Result:
The current weather in New York City is clear with a temperature of 10 degrees Celsius and the humidity is at 50%.
```